



&



信息检索及倒排技术

宋老师
2020/08/30

提纲



- 信息检索概述
- 布尔检索及倒排索引
- 信息检索模型综述
- **索引压缩与VB编码**
- 查询优化与WAND算法
- 信息检索系统评估

NeuHub
AI Links All

为什么要压缩

- 节省磁盘空间
 - 省钱
- 提高内存的利用率
 - 提高速度
- 加快数据从磁盘到内存的传输速度
 - [读取压缩数据][解压缩]比直接[读取未压缩的数据]快
 - 前提：解压缩算法要很快
 - 我们目前所用的解压缩算法在现代硬件上运行相当快

为什么要压缩倒排索引

- 词典
 - 压缩的足够小以便能够放入内存中
 - 当词典足够小时，我们也可以在内存中存储一部分的倒排记录表
- 倒排记录文件
 - 减少所需的磁盘空间
 - 减少从磁盘读取倒排记录文件所需的时间
 - 大的搜索引擎在内存中存储了很大一部分的倒排记录表
 - 压缩可以让我们在内存中存储的更多
- 我们将设计各种基于IR系统的压缩架构

AI Links All

索引|压缩

无损 vs. 有损压缩

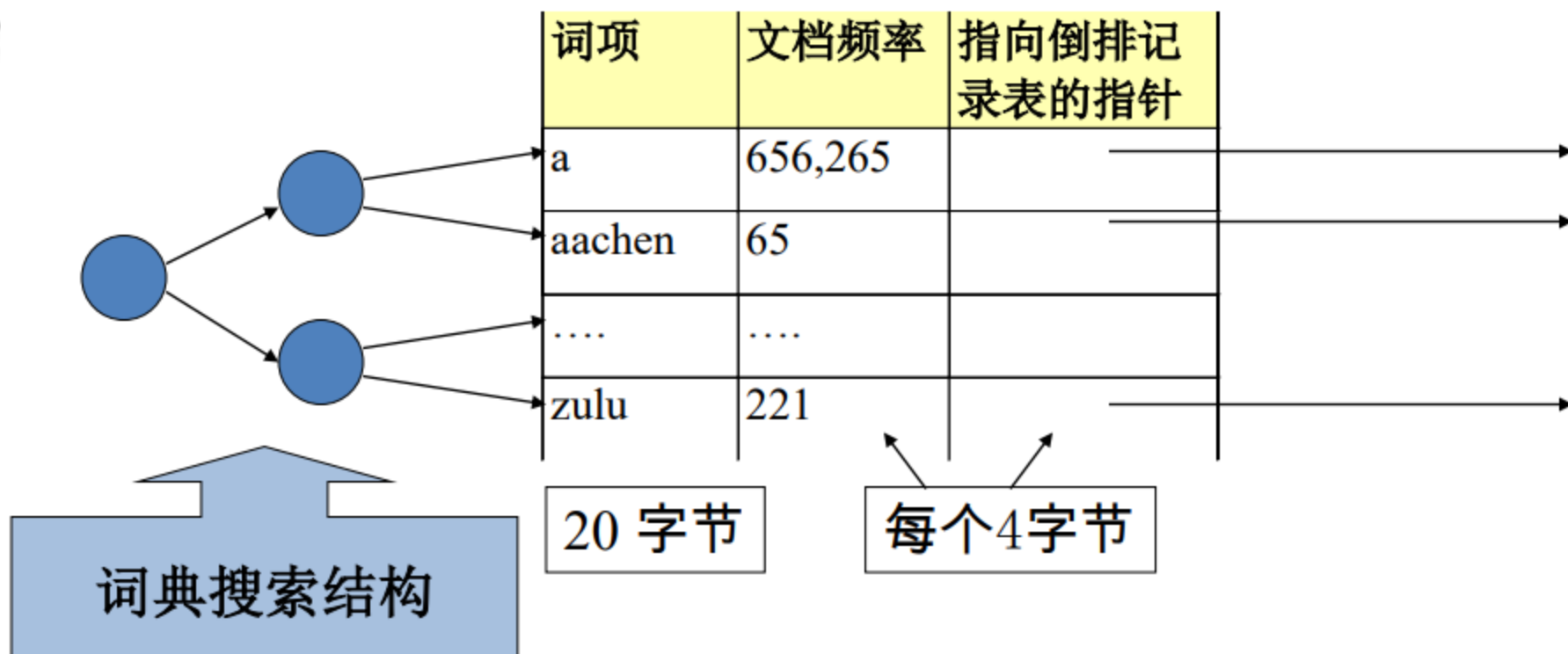
- 无损压缩：压缩之后所有原始信息都被保留。
 - 在IR系统中常采用无损压缩
- 有损压缩：丢掉一些信息
- 一些预处理步骤可以看成是有损压缩：大小写转换，停用词剔除，词干还原，数字去除。
- 第7章：那些削减的倒排记录项都不太可能在查询结果的前k个列表中出现。
 - 对于前k个返回结果来说，这几乎是无损的

有损还是无损与需求相关！！

词典存储

- 定长数组存储

- 400,000词项; 28字节/词项 = 11.2 MB



Links All

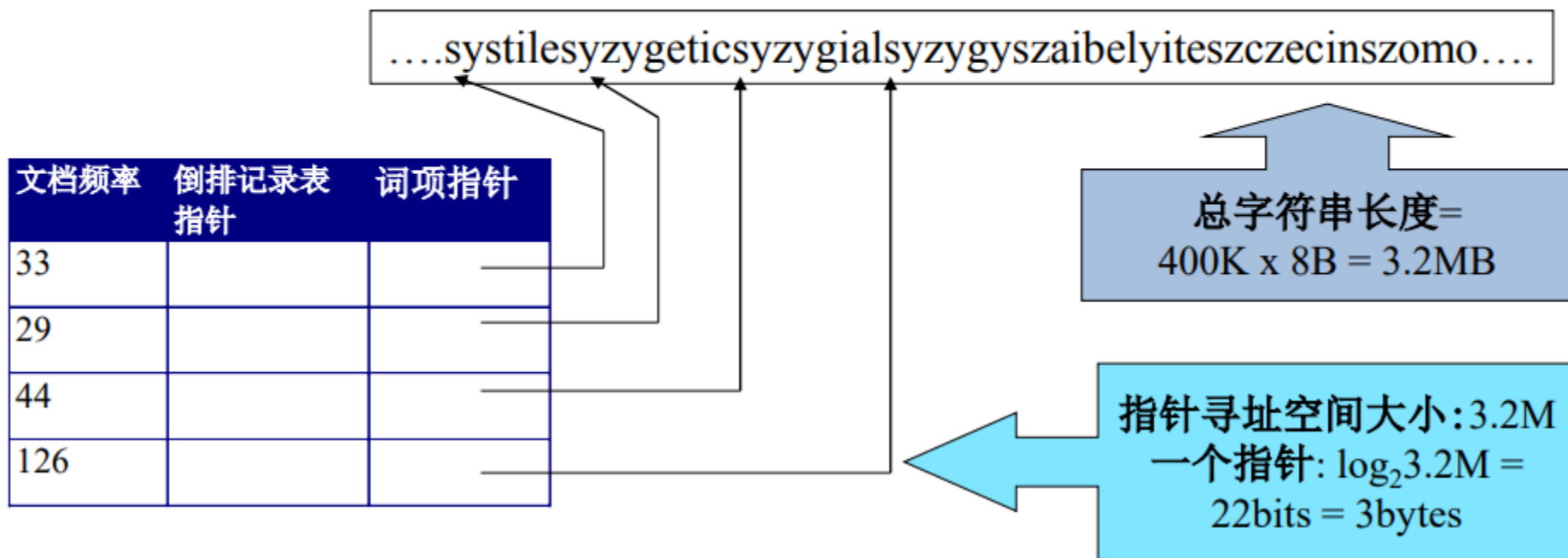
定长方法存储词项浪费空间

- 在词项那一列大部分的字节都被浪费 — 我们为每个词项分配了20字节的固定长度。
 - 但我们仍然不能解决 “*supercalifragilisticexpialidocious*” 和 “*hydrochlorofluorocarbons*”
- 书面英文中单词的平均长度约为4.5个字符
 - 练习：为什么不用这个值来估计词典的大小？
- 英语中平均的词典词项长度为8个字符
 - 平均会有12个字符的空间浪费
- 较短的词项支配了词条的数目但是并不是典型的平均值

Links All

压缩词项列表：将词典看成单一字符串

- 将词典存储为一个长字符串：
 - 指向下一词项的指针同时也标识着当前词项的结束
 - 期望节省60%的词典空间



字符串词典的空间大小

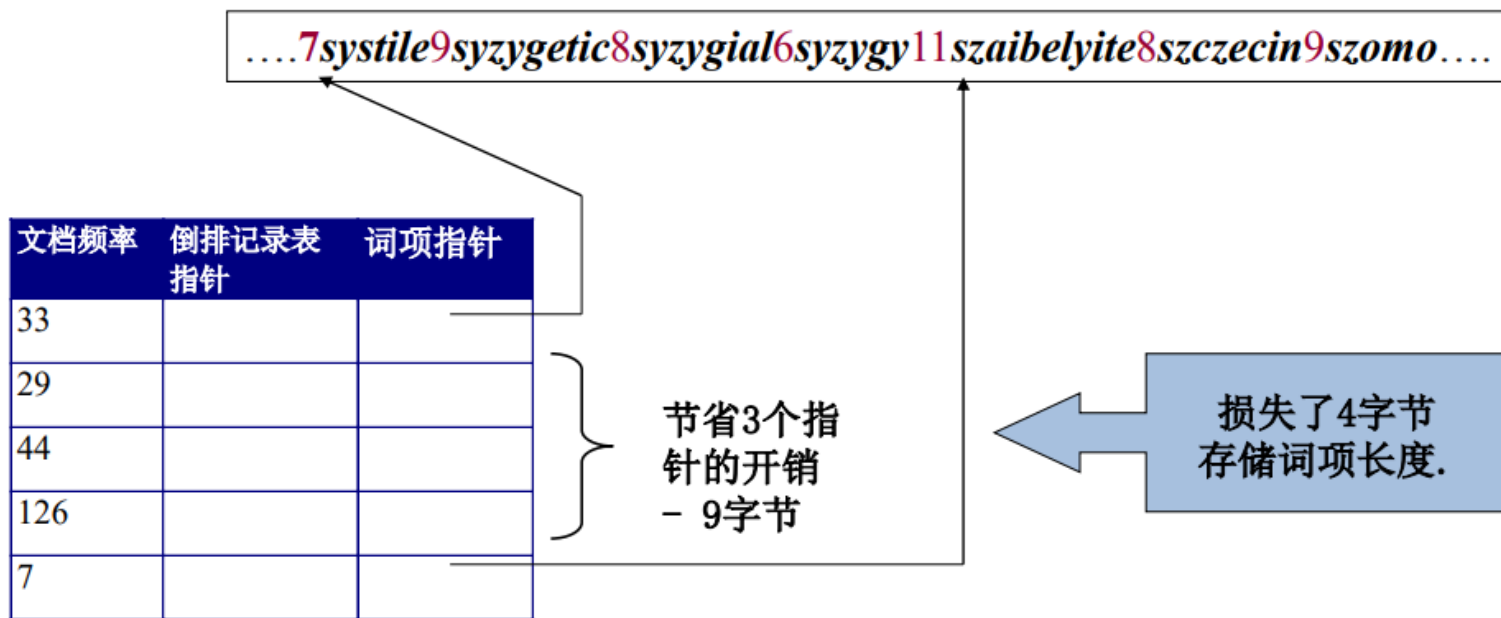
- 词项文档频率 - 4字节
- 倒排记录表指针 - 4字节
- 词项指针 - 3字节

现在是每个词项
平均占用11字节
而不是20字节

- 在词项字符串中，每个词项平均8个字节
- $400k \text{ 词项} \times 19 = 7.6MB$ (固定长度时为11.2MB)

按块存储

- 我们为每第k个词项字符串存储一个指针
 - 下面的例子: $k=4$
- 需要存储词项长度(额外1字节)



节省空间

- 我们取块大小 $k = 4$
- 不采用按块存储时，每个指针花费3字节
 - $3 \times 4 = 12$ 字节

现在我们花费 $3 + 4 = 7$ 字节

削减了另外的0.5MB空间。原来的7.6MB存储空间可以进一步降低到7.1MB

词典压缩小结

数据结构	压缩后大小(MB)
定长数组	11.2
长字符串+词项指针	7.6
按块存储，k=4	7.1
按块存储+前端编码	5.9

索引|压缩

倒排记录表压缩

- 倒排记录表远大于词典，至少为10倍[?]
- 迫切要求：紧密地存储每一个倒排记录表
- 每个倒排记录用文档ID来定义
- 对Reuters (800,000文档)来说，当使用4字节整数表示时，每个文档ID需要32bit
- 或者，我们可以用 $\log_2 800,000 \approx 20 \text{ bits}$ 来表示每个文档ID
- 我们的目标：用远小于20bit来表示每个文档ID

$$100M * 20 / 8 = 250MB$$

索引|压缩

倒排记录表：相反的两点

- 像 “*arachnocentric*” 这样的词项可能在一百万个文档中才会出现一次 - 我们可以用 $\log_2 1M \sim 20$ bits 来存储这一倒排记录。
- 像 “*the*” 这样的词项在每个文档中都会出现，所以对它采用 20bit/倒排记录太浪费了
 - 这种情况更希望是 0/1 的 bit 向量

索引|压缩

规律的探寻：倒排记录表项中文档ID的间距（GAP）

- 我们按照文档ID的递增顺序来存储一个词项的倒排列表。
 - *Computer*: 33, 47, 154, 159, 202...
- 结论：可以存储间距
 - 33, 14, 107, 5, 43...
- 期望：绝大多数间距存储空间都远小于20bit

Links All

索引压缩

找找GAP：3个倒排记录表项

	encoding	postings list				
THE	docIDs	...	283042	283043	283044	283045 ...
	gaps			1	1	1 ...
COMPUTER	docIDs	...	283047	283154	283159	283202 ...
	gaps			107	5	43 ...
ARACHNOCENTRIC	docIDs	252000	500100			
	gaps	252000	248100			

索引|压缩

可变长度编码

- 目标:
 - 对于 *arachnocratic*, 我们使用20bit/间距项
 - 对于 *the*, 我们使用1bit/间距项
- 如果一个词项的平均间距为G, 我们想使用 $\log_2 G$ bit/间距项
- 关键问题: 需要利用整数个字节来对每个间距编码
- 这需要一个可变长度编码
- 可变长度编码对一些小数字使用短码来实现这一点

索引|压缩

GAP→可变字节码 (Variable Byte)

- 对一个间距值 G ，我们想用最少的所需字节来表示 $\log_2 G$ bit
- 先用一个字节来存储 G ，并分配1bit作为延续位 c
- 如果 $G \leq 127$ ，对7位有效码采用二进制编码并设延续位 $c=1$
- 若 $G > 127$ ，则先对 G 低阶的7位编码，然后采取相同的算法用额外的字节对高阶bit位进行编码
- 设置最后一个字节的延续位为1 ($c=1$)，其他字节的 $c=0$

索引|压缩

Compress number 512312 or 1111101000100111000 in binary.

- How many bytes? 11111|0100010|0111000. 3 bytes!
- How do we compress the number?
 1. Extract the lowest 7 bits.
 $512312 \bmod 128 = 56 = 0111000$
 2. Discard lowest 7 bits.
 $512312 \div 128 = 4002$ (or $512312 >> 7$)
 3. Extract the lowest 7 bits.
 $4002 \bmod 128 = 34 = 0100010$
 4. Discard lowest 7 bits.
 $4002 \div 128 = 31$ (or $4002 >> 7$)
 5. Number smaller than 128. Write in lowest 7 bits and set top bit to 1. $31 = 11111$ So we write 10011111 which is
 $31 + 128 = 159$

sub
AI Links All

索引|压缩

Encoding

```
1: function ENCODE( $x$ )  
2:   while  $x \geq 128$  do  
3:     WRITE( $x \bmod 128$ )  
4:      $x = x \div 128$   
5:   end while  
6:   WRITE( $x + 128$ )  
7: end function
```

Decoding

```
1: function DECODE(bytes)  
2:    $x = 0, s = 0$   
3:    $y = \text{READBYTE}(\text{bytes})$   
4:   while  $y < 128$  do  
5:      $x = x \wedge (y \ll s)$   
6:      $s = s + 7$   
7:      $y = \text{READBYTE}(\text{bytes})$   
8:   end while  
9:    $x = x \wedge ((y - 128) \ll s)$   
10:  return  $x$   
11: end function
```

索引|压缩

RCV1 压缩

数据结构	Size in MB
词典, 定长数组	11.2
词典, 长字符串+词项指针	7.6
词典, 按块存储, $k = 4$	7.1
词典, 按块存储+前端编码	5.9
文档集(文本、XML标签等)	3,600.0
文档集(文本)	960.0
词项关联矩阵	40,000.0
倒排记录表, 未压缩(32位字)	400.0
倒排记录表, 未压缩(20bit)	250.0
倒排记录表, 可变字节码	116.0
倒排记录表, γ 编码	101.0

AI Links All

提纲

- 信息检索概述
- 布尔检索及倒排索引
- 信息检索模型综述
- 索引压缩与VB编码
- **查询优化与WAND算法**
- 信息检索系统评估

NeuHub
AI Links All

查询处理

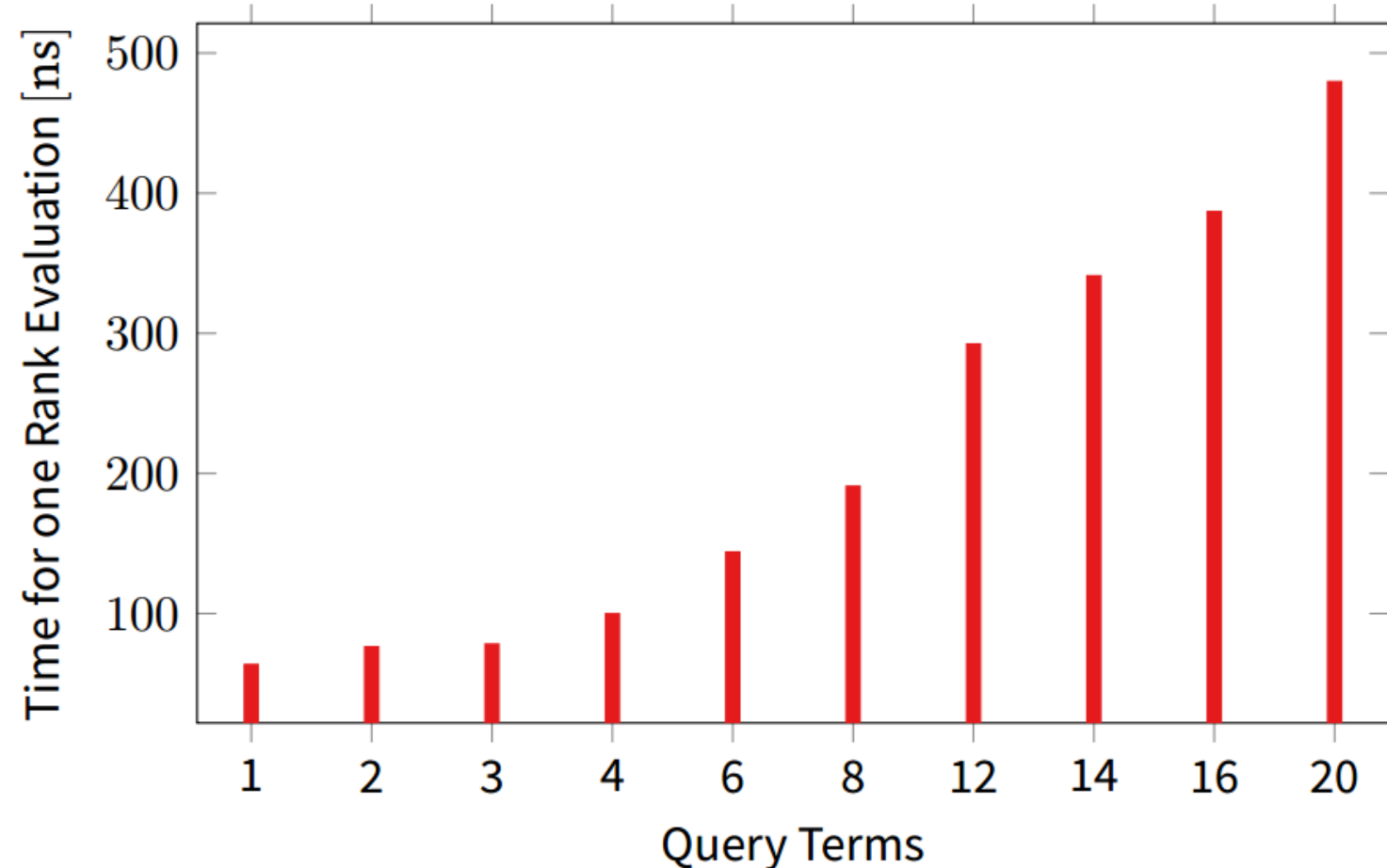
BM25 computation for one document:

$$S_{Q,d}^{BM25} = \sum_{q \in Q} \frac{(k_1 + 1)f_{d,q}}{k_1 \left(1 - b + b \frac{n_d}{n_{avg}}\right) + f_{d,q}} \cdot \ln \left(\frac{N - F_{\mathcal{D},q} + 0.5}{F_{\mathcal{D},q} + 0.5} \right)$$

Symbol	Description
Q	Multi set of query terms
q	Query term in Q
d	Evaluated document
$f_{d,q}$	Frequency of q in d (Term frequency)
N	Number of documents in the collection
$F_{\mathcal{D},q}$	Number of documents containing q (Document frequency)
n_d	Length of document d
n_{avg}	Average document length in the collection
k_1, b	Constants

查询处理

BM25 computation speed:



查询处理

- Evaluating BM25 for one document takes 100 nanoseconds (Fast!).
- Assume query matches 25 million documents (term “the” contained in almost all documents).
- $25 \text{ million} \times 100 \text{ nanoseconds} \approx 2.5 \text{ seconds}$.
- Is waiting 2.5 seconds for a search query acceptable?

查询处理优化

IDEA: Top- k Retrieval

Retrieve the top k items for a given query without having to evaluate all documents.

- Web search engines return only the top-10 results to the user.
- For most queries most users generally do not retrieve more documents.
- No need to score all possible documents to produce a “complete” ranking.

Links All

WAND



Wand 算法通过计算每个词的贡献上限来估计文档的相关性上限，并与预设的阈值比较，进而跳过一些相关性一定达不到要求的文档，从而得到提速的效果。

Wand 算法首先要估计每个词对相关性的贡献上限 (upper bound)，最简单的相关性就是 TF-IDF，一般IDF是固定的，因此只需要估计一个词在各个文档中的词频TF上限(即这个词在各个文档中最大的TF)，该步骤通过线下计算即可完成。

线下计算出各个词的相关性上限，可以计算出一个 query 和一个文档的相关性上限值，就是他们共同出现的词的相关性上限值的和，通过与预设的阈值比较，如果query 与文档的相关性大于阈值，则进行下一步的计算，否则丢弃。

WAND

Query Q: The quick brown fox with $k = 2$

The

2	3	7	8	9	10	11	12	13	17	18	19	...
---	---	---	---	---	----	----	----	----	----	----	----	-----

quick

5	6	9	11	14	18
---	---	---	----	----	----

brown

2	4	5	15	42	84	96
---	---	---	----	----	----	----

fox

5	7	8	13
---	---	---	----

WAND

Query Q: The quick brown fox

with $k = 2$



&



Maximum Contribution for each query term

	Max	
The	0.9	2 3 7 8 9 10 11 12 13 17 18 19 ...
quick	1.9	5 6 9 11 14 18
brown	2.3	2 4 5 15 42 84 96
fox	7.1	5 7 8 13

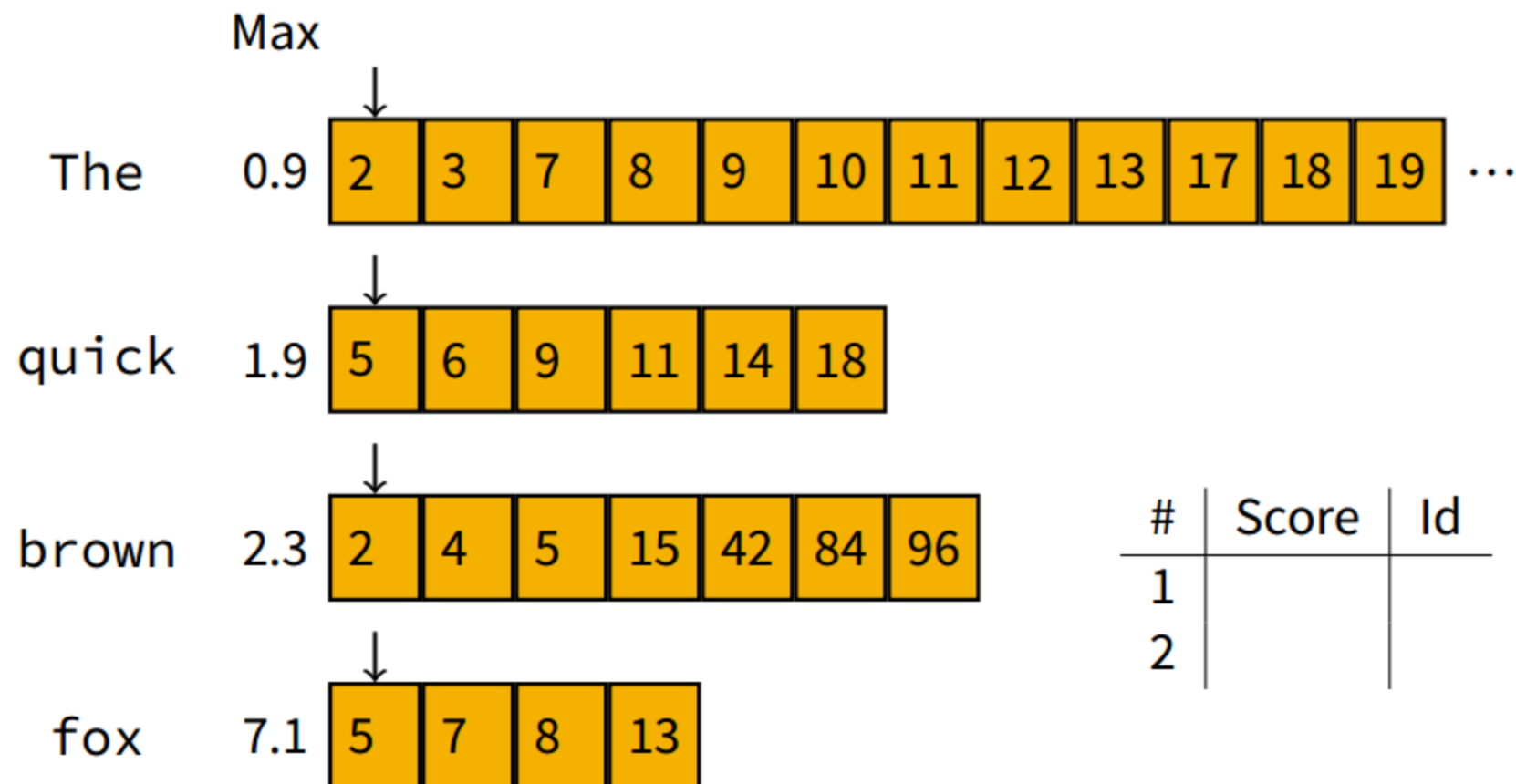
WAND

Query Q: The quick brown fox

with $k = 2$



&



#	Score	Id
1		
2		

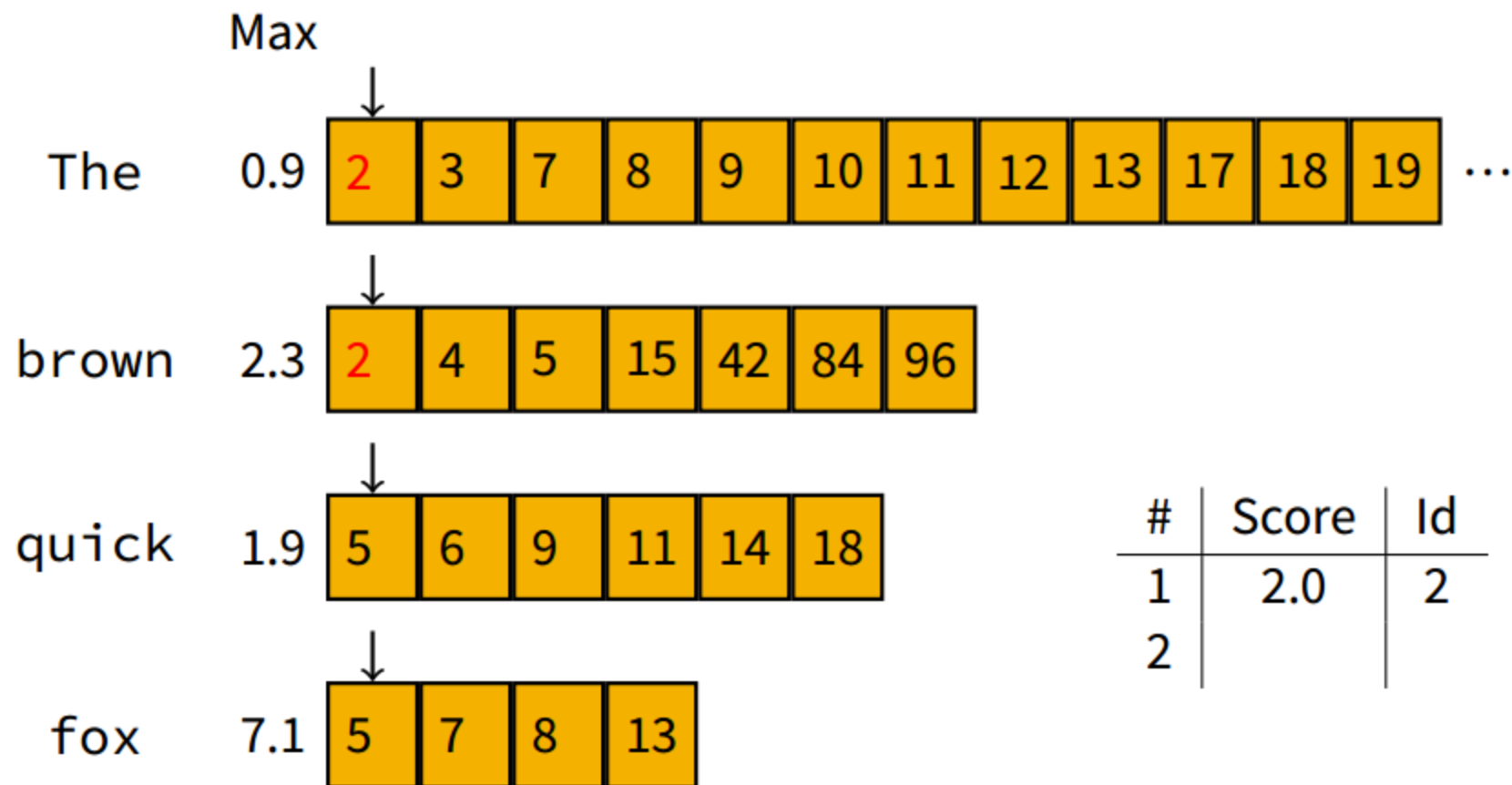
All

WAND

Query Q: The quick brown fox

with $k = 2$

(1) Reorder based on current id and score smallest.

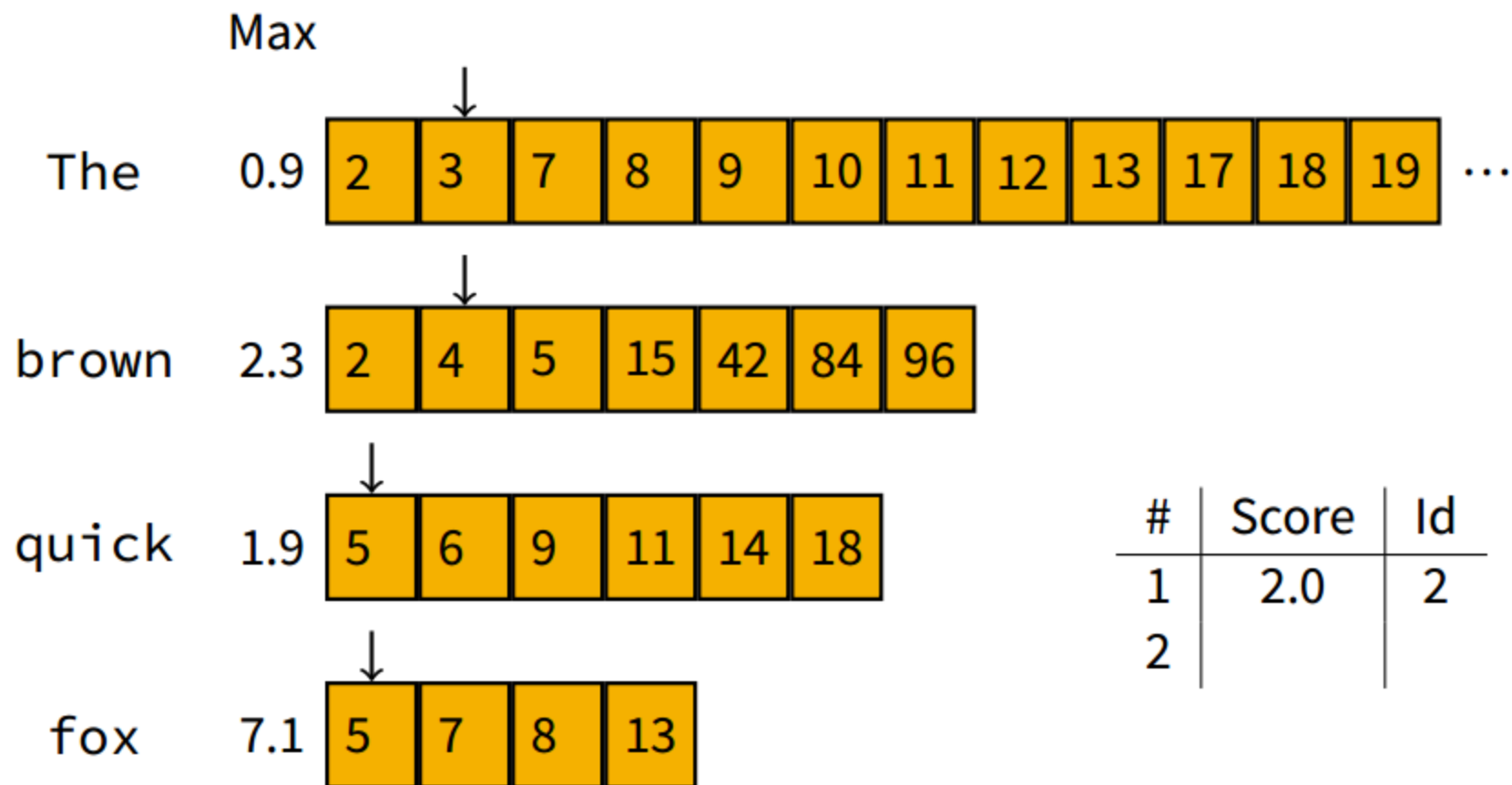


WAND

Query Q: The quick brown fox

with $k = 2$

(2) Move pointer of scored elements.

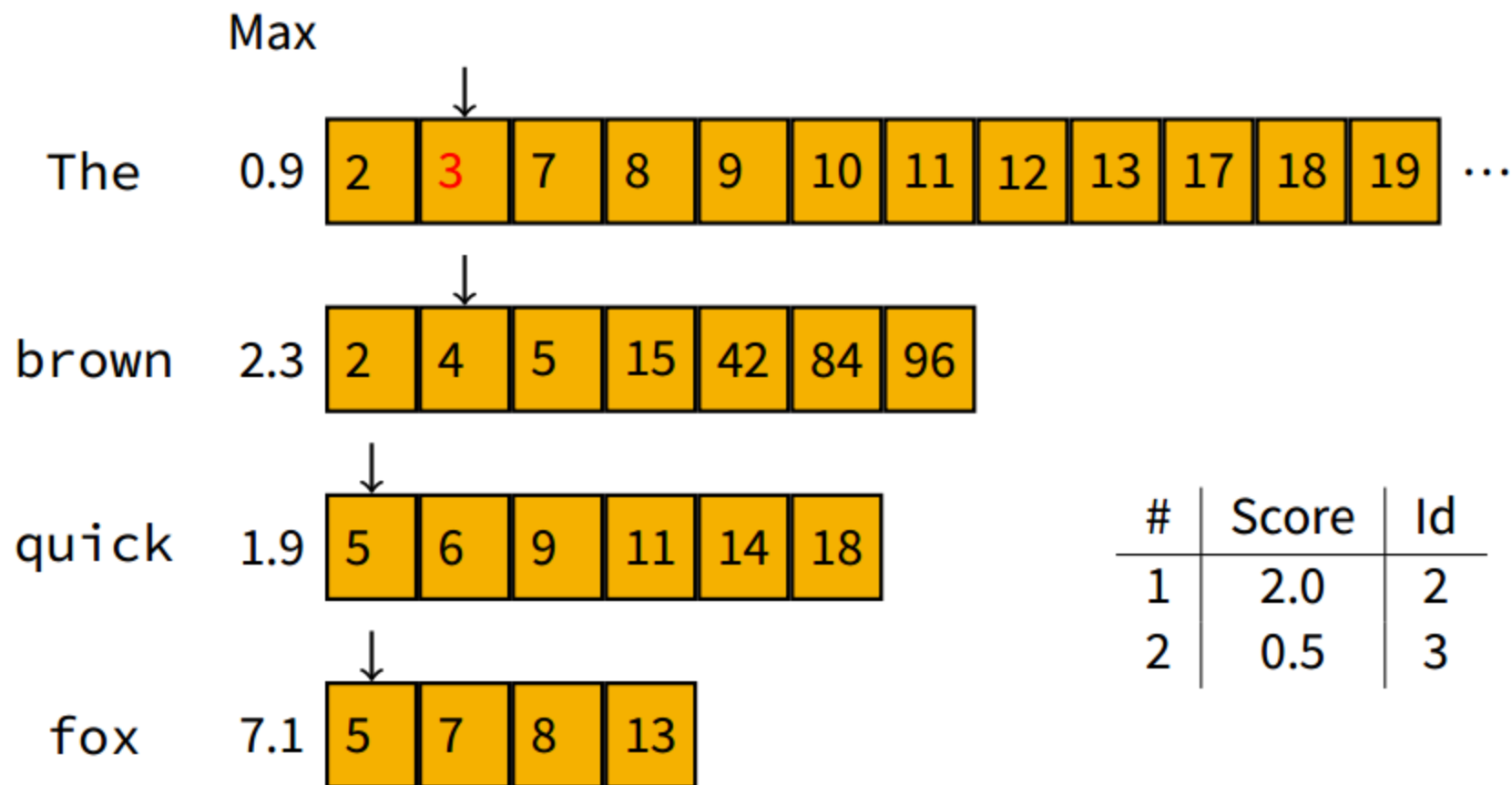


WAND

Query Q: The quick brown fox

with $k = 2$

(3) Reorder based on current id and score smallest.

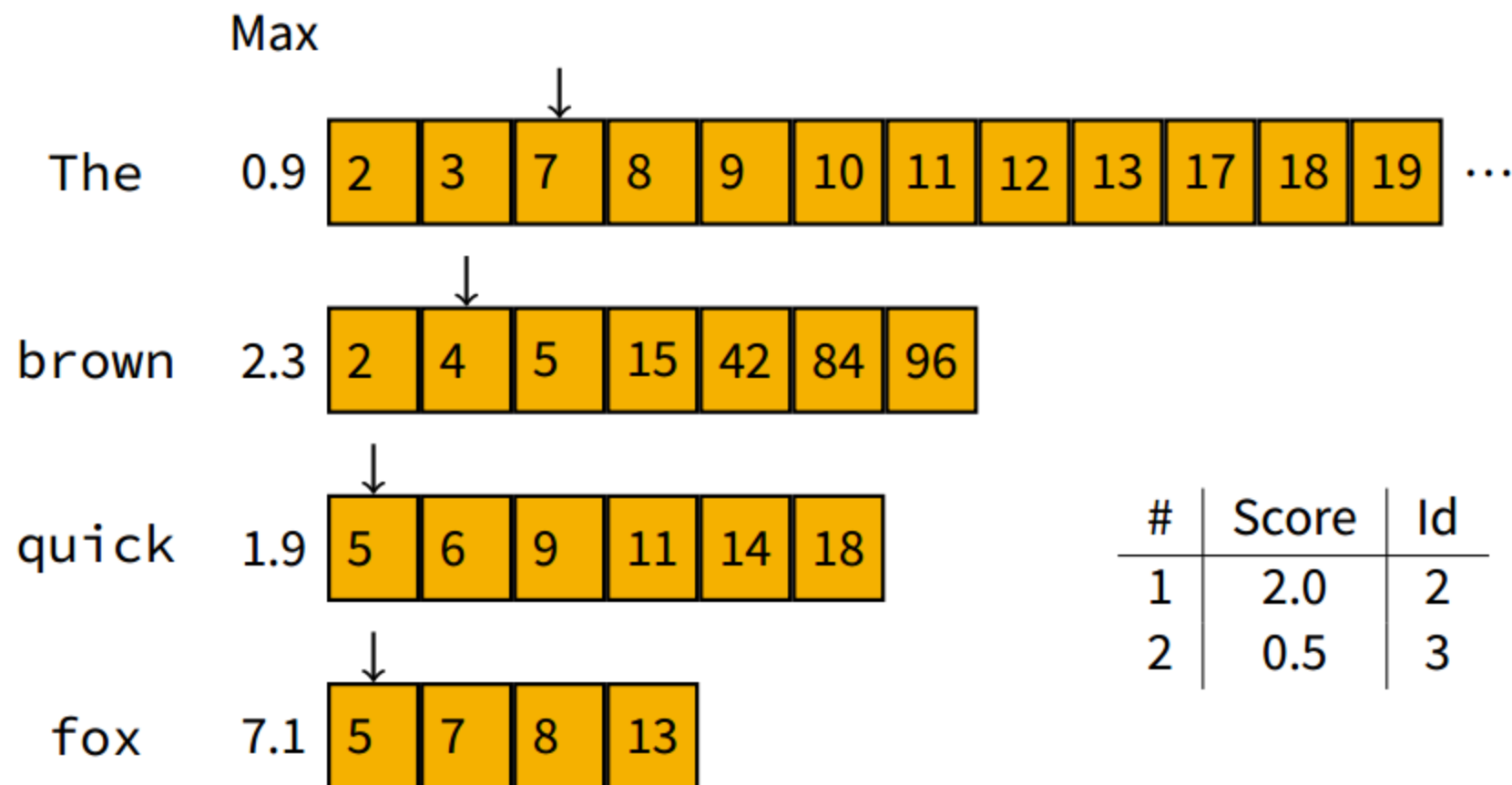


WAND

Query Q: The quick brown fox

with $k = 2$

(4) Move pointer of scored elements





&

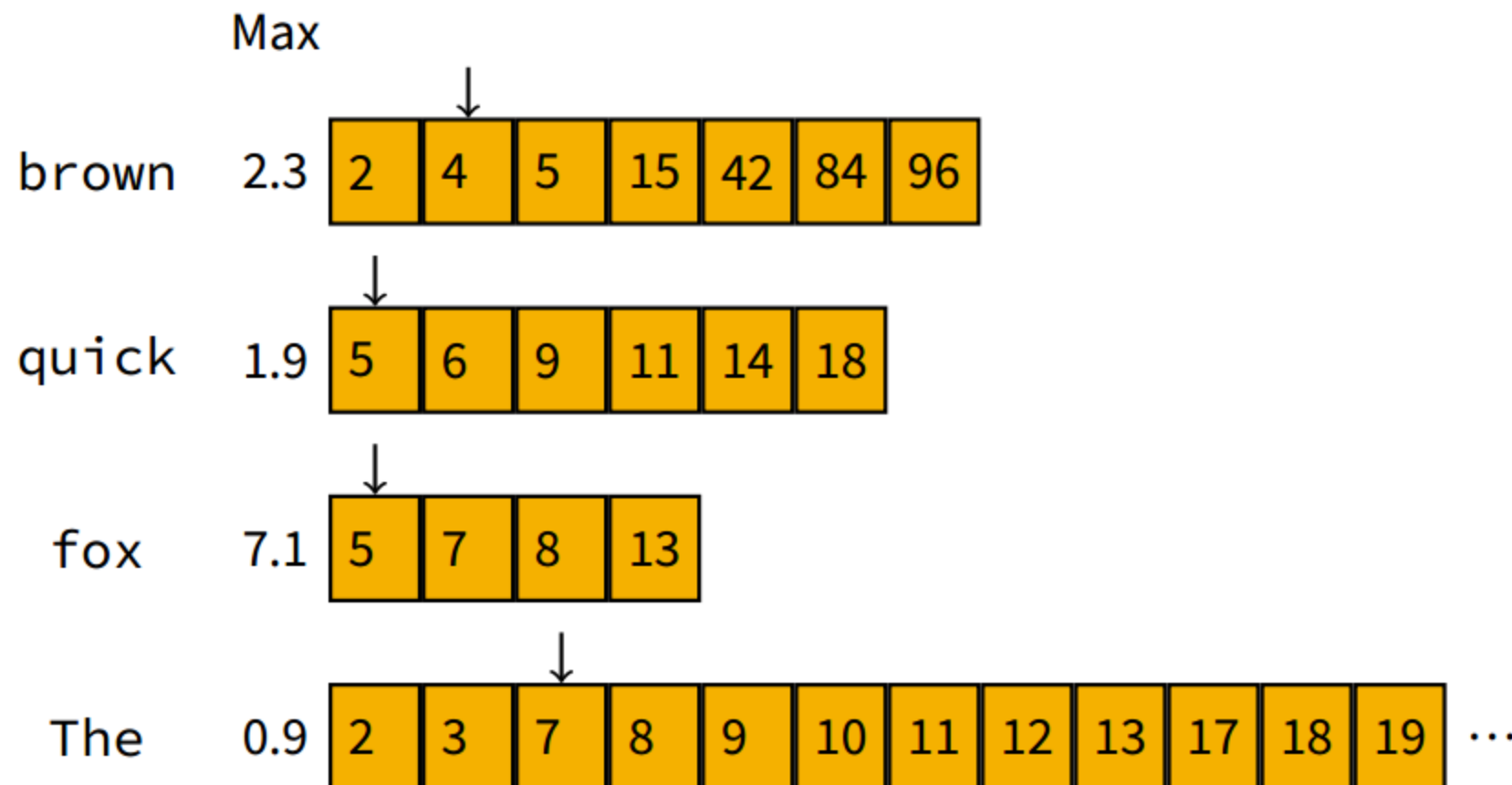


WAND

Query Q: The quick brown fox

with $k = 2$

(5) Reorder based on current id and score smallest.





&

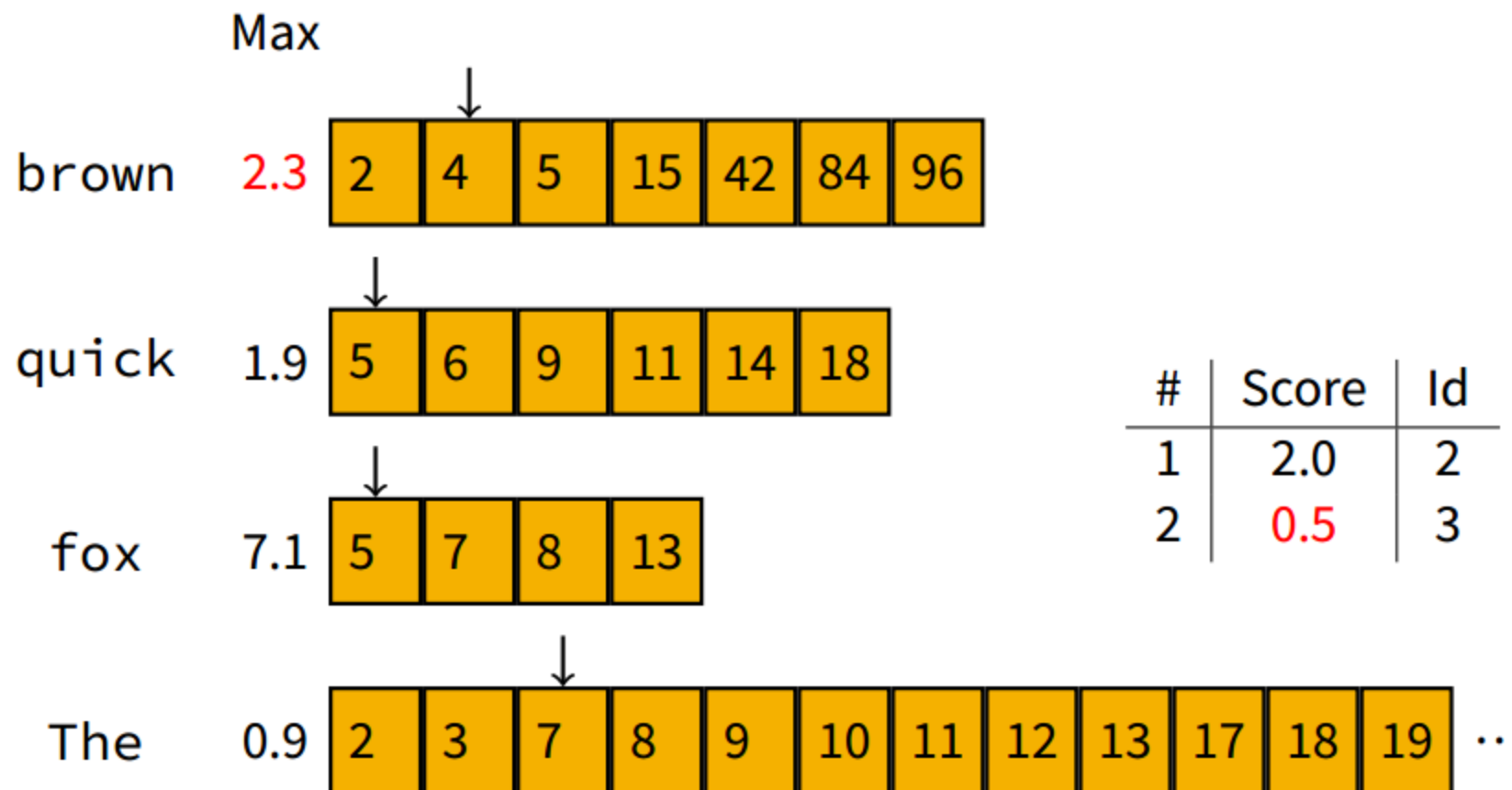


WAND

Query Q: The quick brown fox

with $k = 2$

(6) Decide if we need to score smallest id.

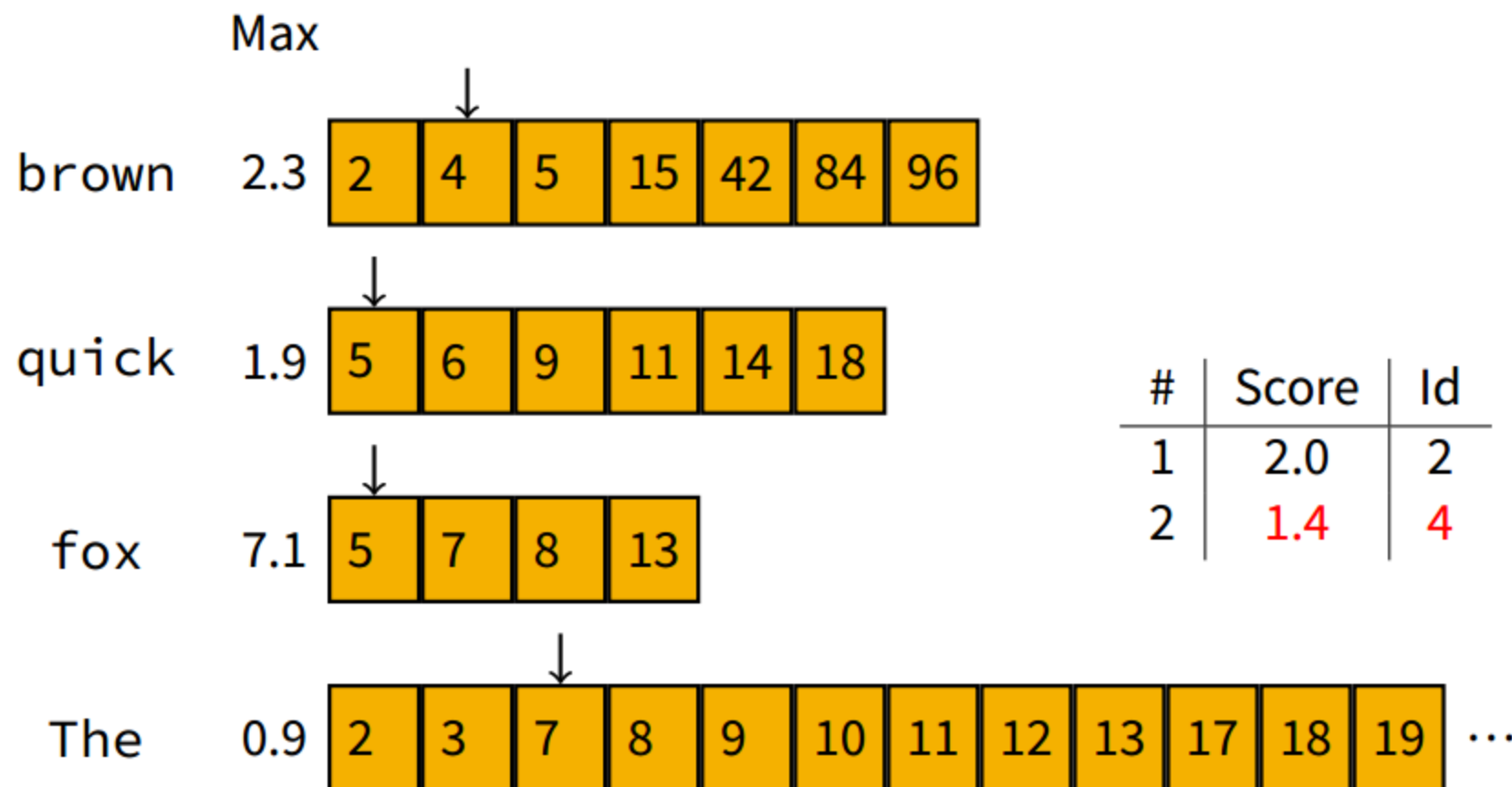


WAND

Query Q: The quick brown fox

with $k = 2$

(7) Replace 3 with 4 on the heap.





&

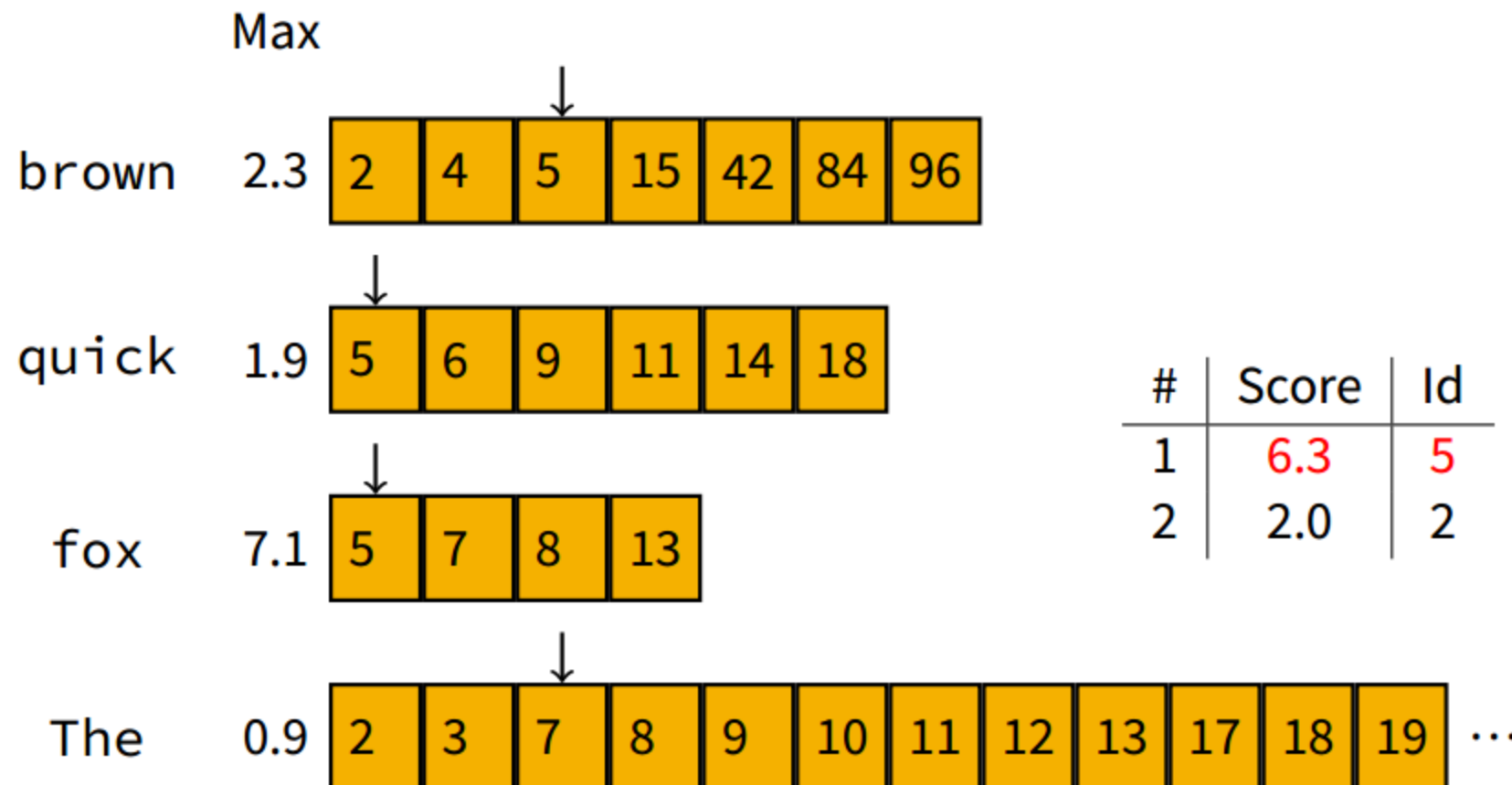


WAND

Query Q: The quick brown fox

with $k = 2$

(8) Sort by current id. Evaluate 5. Add to Heap.

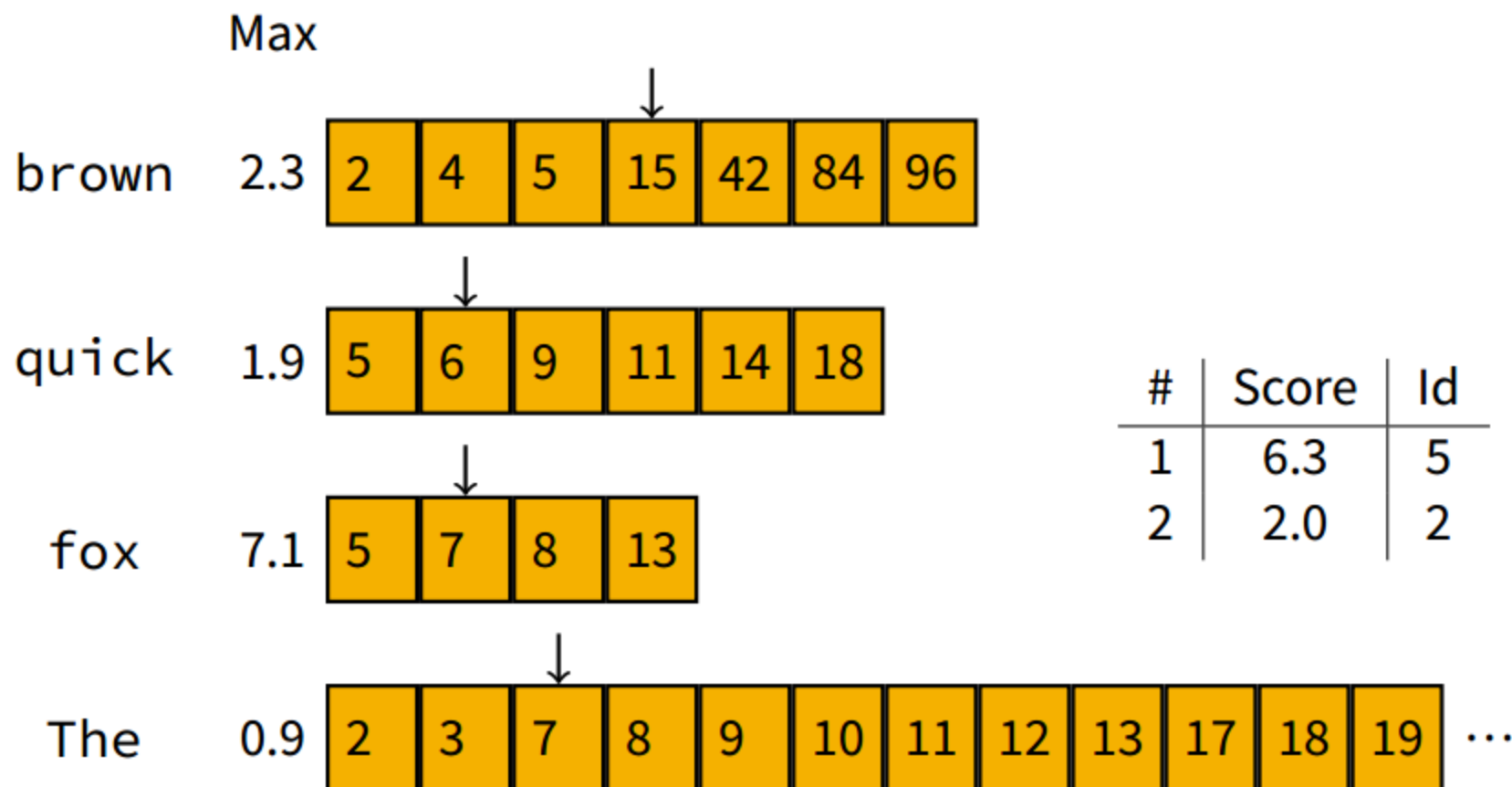


WAND

Query Q: The quick brown fox

with $k = 2$

(9) Move pointers and sort.



WAND

Query Q: The quick brown fox

with $k = 2$



&

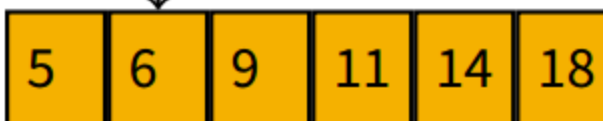


(10) Use max to skip scoring smallest.

Max

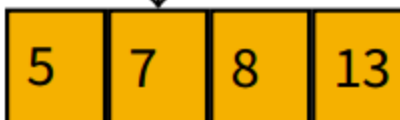
quick

1.9



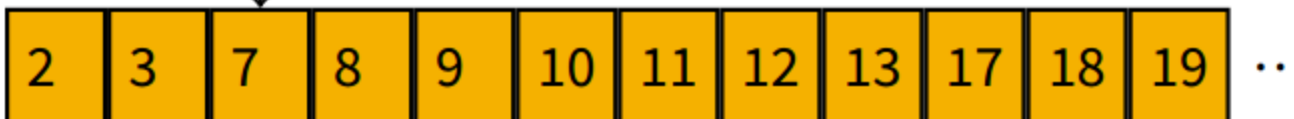
fox

7.1



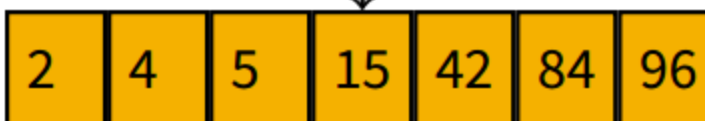
The

0.9



brown

2.3



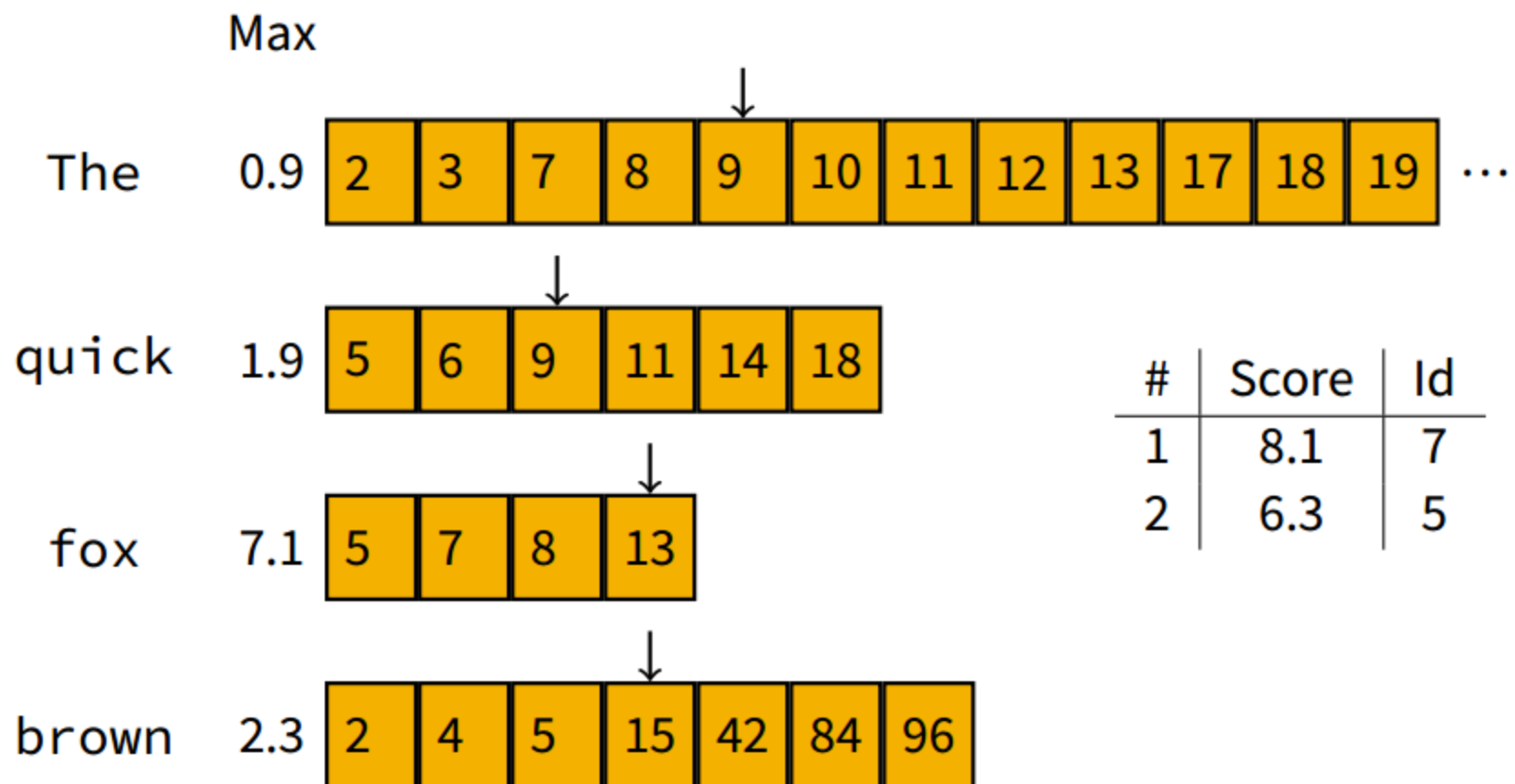
#	Score	Id
1	6.3	5
2	2.0	2

WAND

Query Q: The quick brown fox

with $k = 2$

Do we have to evaluate document 9?

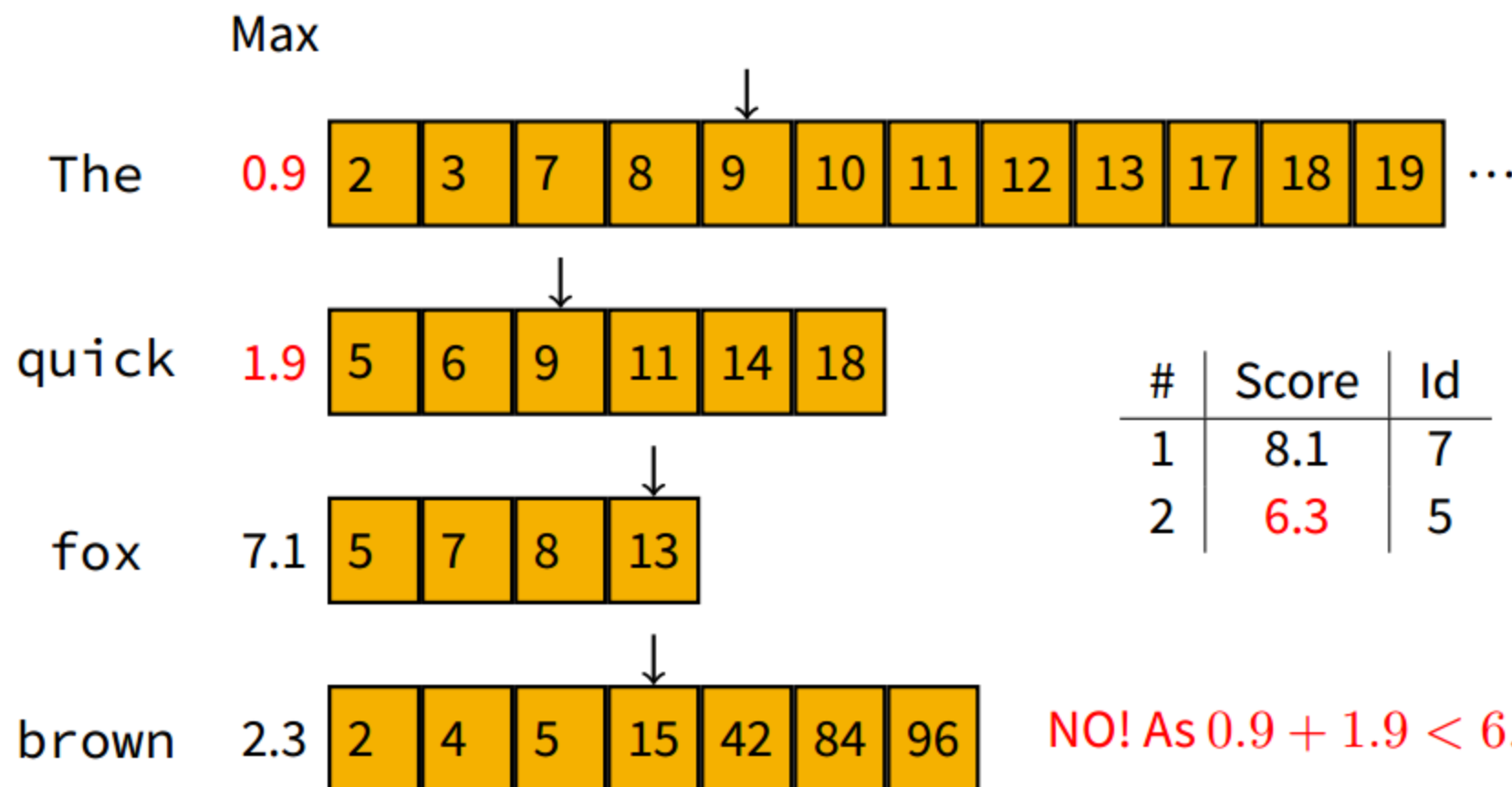


WAND

Query Q: The quick brown fox

with $k = 2$

Do we have to evaluate document 9?

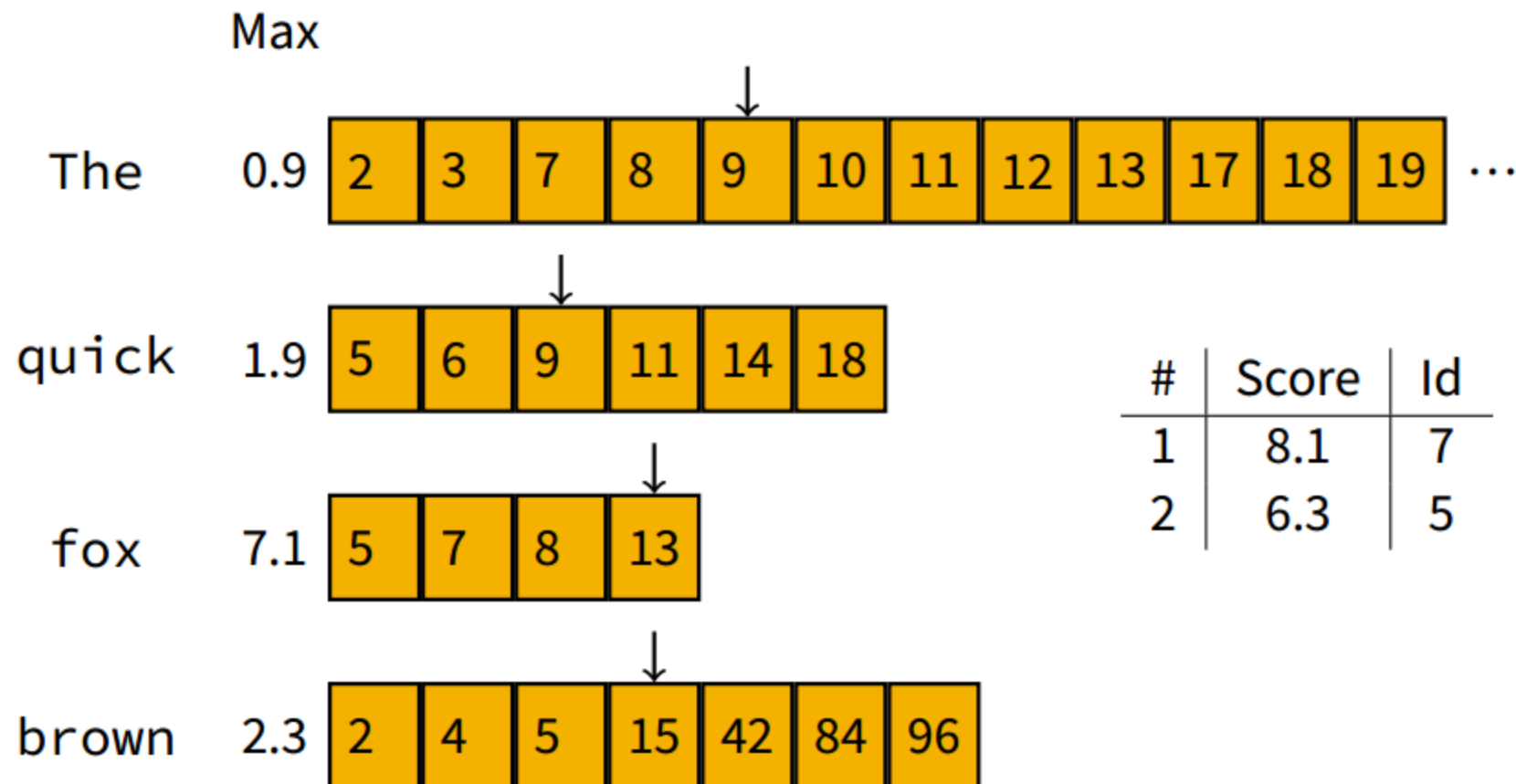


WAND

Query Q: The quick brown fox

with $k = 2$

What is the next document that has to be evaluated?



WAND

Query Q: The quick brown fox

with $k = 2$

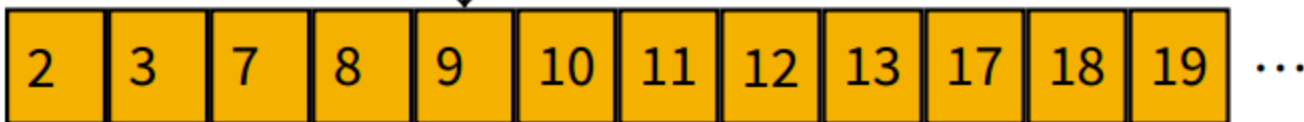


What is the next document that has to be evaluated?

Max

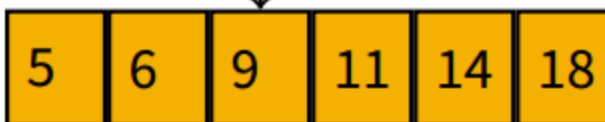
The

0.9



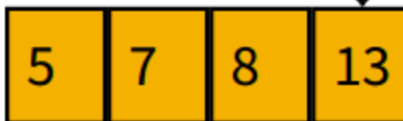
quick

1.9



fox

7.1



brown

2.3



#	Score	Id
1	8.1	7
2	6.3	5

13, as $0.9 + 1.9 + 7.1 > 6.3$!

WAND

Query Q: The quick brown fox

with $k = 2$



&

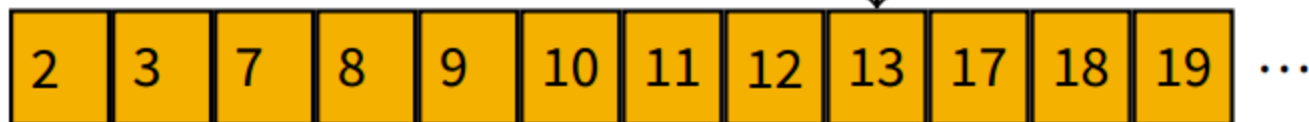


Fast forward smaller ids to 13 (GEQ) and sort.

Max

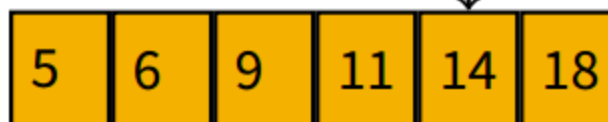
The

0.9



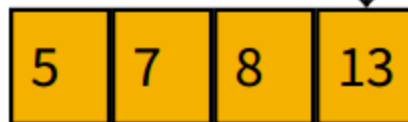
quick

1.9



fox

7.1



brown

2.3



#	Score	Id
1	8.1	7
2	6.3	5

WAND



&



Given Q, k and the postings lists $L[0 \dots |Q| - 1]$ with:

$L[i].max \hat{=}$ the **maximum contribution** of the list

$L[i].cur \hat{=}$ the current element of the list

```
1: function WAND( $Q, k, L[0 \dots |Q| - 1]$ )
2:    $TopDocs = \emptyset$  ▷ Min Heap of size  $k$ 
3:   Threshold  $\Theta = 0$  ▷ Smallest score in  $TopDocs$ 
4:   while Not all lists are processed do
5:     Sort  $L$  based on  $L[i].cur$ 
6:     Select pivot list  $p$  such that  $\sum_{i=0}^{p-1} L[i].max \geq \Theta$ 
7:     Forward all lists  $L[0 \dots |p| - 1]$  to  $d_p = L[p].cur$ 
8:     Compute  $S_{Q, d_p}^{BM25}$  and insert into  $TopDocs$  if score  $> \Theta$ 
9:      $\Theta = \min(TopDocs)$  or 0 if  $|TopDocs| < k$ .
10:  end while
11:  Return  $TopDocs$ 
12: end function
```

Links All

WAND

Method	$k = 10$		$k = 1,000$	
	Avg.	Med.	Avg.	Med.
Exhaustive	100.0	100.0	100.0	100.0
BM25	3.5	1.0	28.0	11.7

Fraction of pointers processed as a percentage of the total number of pointers associated with each query, GOV2, using TREC topics 701–850. Across the set of queries, the average number of postings per query for exhaustive processing is 1,460,562, and the median number of postings is 1,080,008. The percentages shown in the table are relative to these two numbers.

Links All

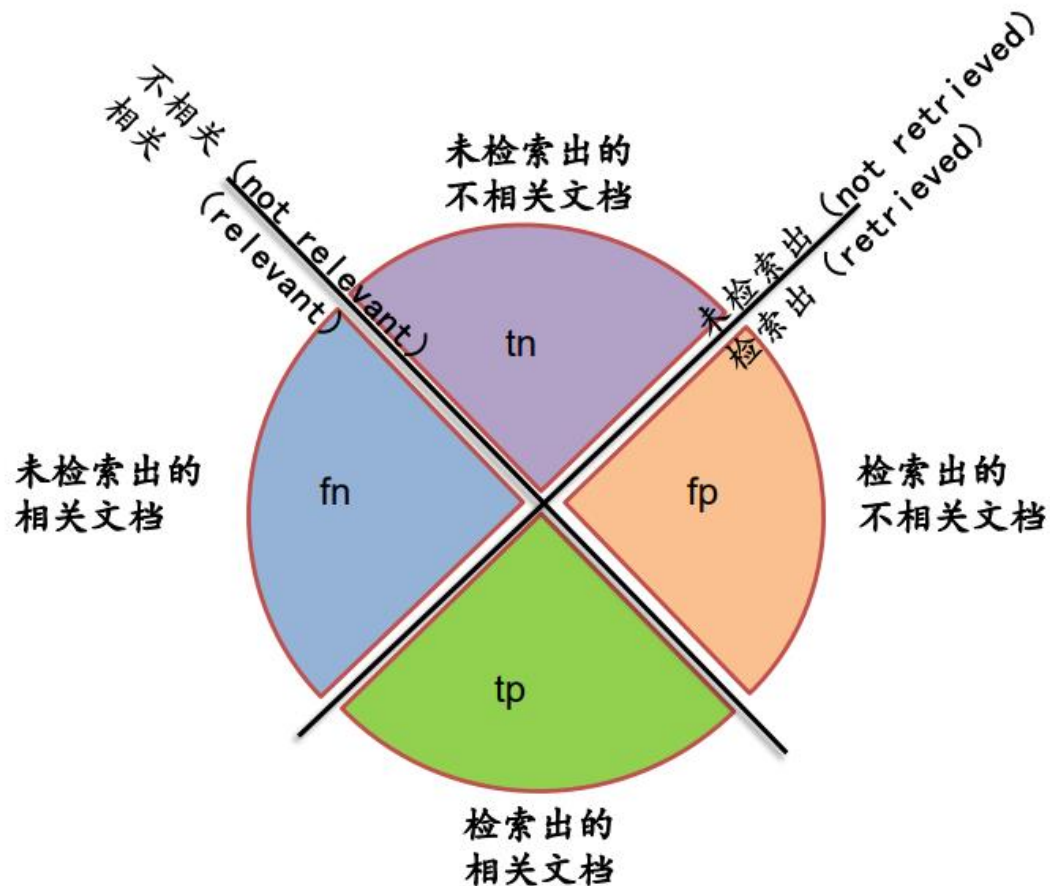
提纲

- 信息检索概述
- 布尔检索及倒排索引
- 信息检索模型综述
- 索引压缩与VB编码
- 查询优化与WAND算法
- **信息检索系统评估**

NeuHub
AI Links All

检索的评价

按照文档“是否相关” “是否被检索出” 划分



检索的评价

与排序无关的检索评价：正确率和召回率

- **正确率/查准率**：返回的相关文档占返回文档总数的百分比
- **召回率/查全率**：返回的相关文档占所有相关文档的百分比。

	Relevant	Nonrelevant
Retrieved	真正例 (true positives, tp)	伪正例 (false positives, fp)
Not Retrieved	伪反例 (false negatives, fn)	真反例 (true negatives, tn)

正确率/查准率**Precision**

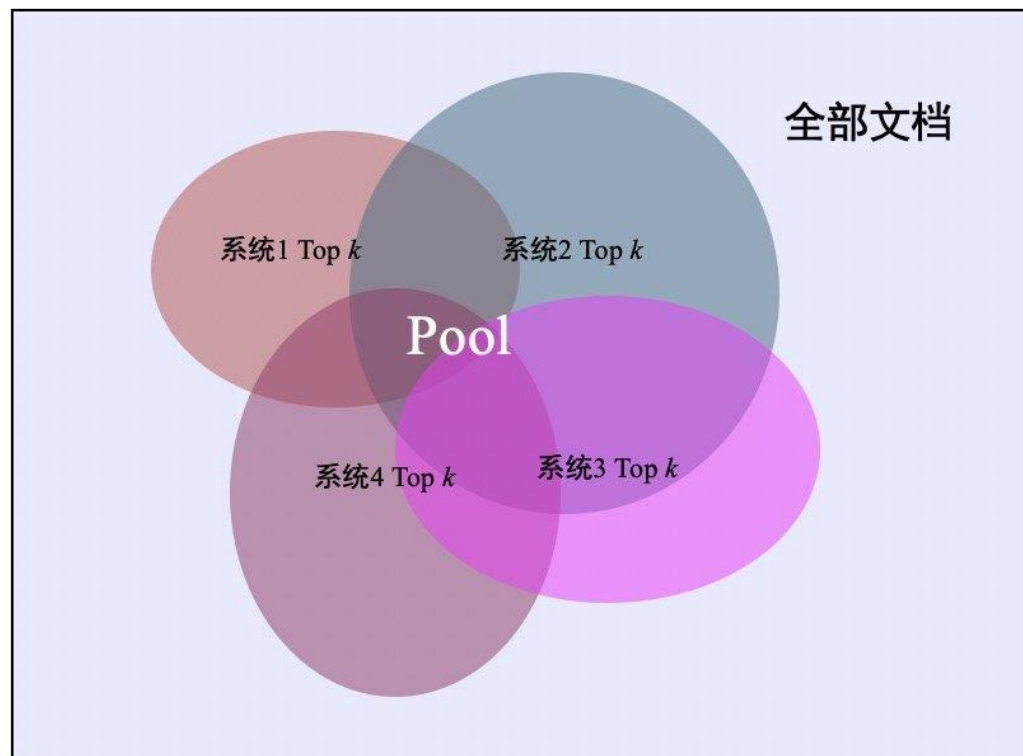
召回率/查全率**Recall**

$P = tp / (tp + fp)$

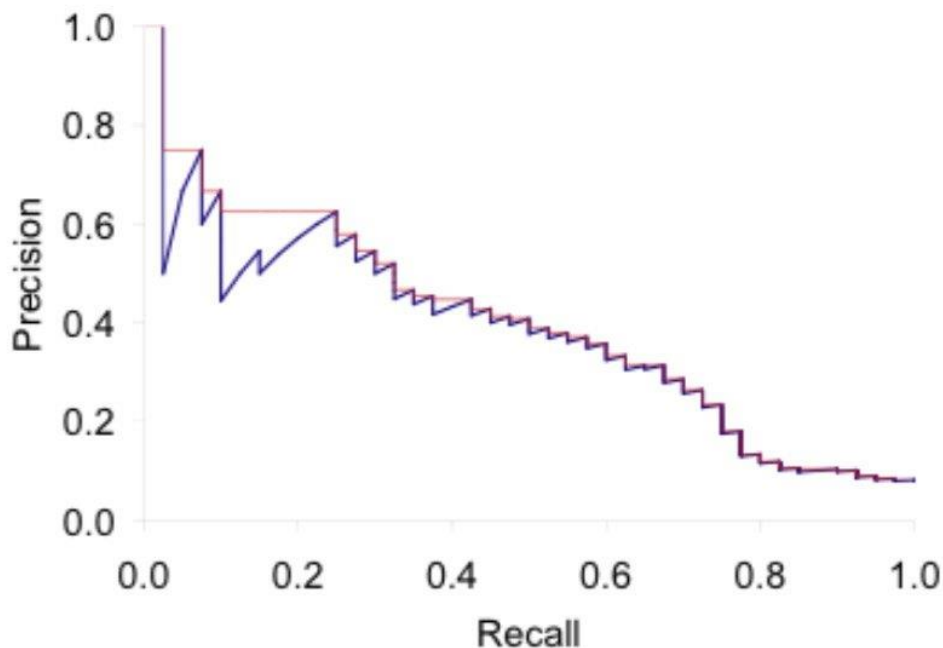
$R = tp / (tp + fn)$

Pooling方法计算召回率

4个系统的Pooling



检索的评价



- 插值 (红色): 从观察到的值, 得到未观察数据的值, 将来所有点上的最高结果
- 插值后可以得到一个较平滑的曲线, 来去掉细小变化的影响, 更合理的评价搜索引擎。

检索的评价

更多的评价准则：AP

- 平均正确率 (Average Precision, AP): 对不同召回率点上的正确率进行平均
 - **未插值的AP:** 某个查询Q共有6个相关结果, 某系统排序返回了5篇相关文档, 其位置分别是第1, 第2, 第5, 第10, 第20位, 则 $AP = (1/1 + 2/2 + 3/5 + 4/10 + 5/20 + 0)/6$
 - **插值的AP:** 在召回率分别为0, 0.1, 0.2, ..., 1.0的十一个点上的正确率求平均, 等价于11点平均
 - **只对返回的相关文档进行计算的AP**
 $AP = (1/1 + 2/2 + 3/5 + 4/10 + 5/20)/5$, 倾向那些快速返回结果的系统, 没有考虑召回率

AI Links All

检索的评价-Precision@N

回到例子

系统&查询	1	2	3	4	5
系统1, 查询1	d3 ✓	d6 ✓	d8	d10	d11
系统1, 查询2	d1 ✓	d4	d7	d11	d13 ✓
系统2, 查询1	d6 ✓	d7	d2	d9 ✓	/
系统2, 查询2	d1 ✓	d2 ✓	d4	d13 ✓	d14

系统1查询1: $P@2=1$, $P@5=2/5$; 系统1查询2: $P@2=1/2$, $P@5=2/5$;
 系统2查询1: $P@2=1/2$, $P@5=2/5$; 系统2查询2: $P@2=1$, $P@5=3/5$

检索的评价

更多的评价准则：MAP

- **平均正确率均值 Mean Average Precision (MAP)**

- 在每个相关文档位置上正确率的平均值，被称为平均正确率 (AP)
- 对所有查询求平均，就得到平均正确率均值 (MAP)

$$\text{MAP}(Q) = \frac{1}{|Q|} \sum_{j=1}^{|Q|} \frac{1}{m_j} \sum_{k=1}^{m_j} \text{Precision}(R_{jk})$$

- **参数说明**

- Q 为信息需求， $q_j \in Q$ 所对应的所有相关文档集合为 $\{d_1, d_2, \dots, d_{m_j}\}$ ， R_{jk} 是查询 q_j 的返回结果、该结果中包含 $\{d_1, d_2, \dots, d_k\}$ 而不含有 d_{k+1} 及以后的相关文档

检索的评价

回到例子

系统&查询	1	2	3	4	5
系统1, 查询1	d3 ✓	d6 ✓	d8	d10	d11
系统1, 查询2	d1 ✓	d4	d7	d11	d13 ✓
系统2, 查询1	d6 ✓	d7	d2	d9 ✓	/
系统2, 查询2	d1 ✓	d2 ✓	d4	d13 ✓	d14

系统1查询1: $P=2/5, R=2/4, F=4/9, AP=1/2$; 系统1查询2: $P=2/5, R=2/3, F=1/2, AP=7/15$;

系统2查询1: $P=2/4, R=2/4, F=1/2, AP=3/8$; 系统2查询2: $P=3/5, R=3/3, F=3/4, AP=11/12$

系统1的 $\text{MacroP}=2/5, \text{MacroR}=7/12, \text{MacroF}=17/36, \text{MAP}=29/60, \text{MicroP}=4/10, \text{MicroR}=4/7, \text{MicroF}=8/17$

系统2的 $\text{MacroP}=11/20, \text{MacroR}=3/4, \text{MacroF}=5/8, \text{MAP}=31/48, \text{MicroP}=5/9, \text{MicroR}=5/7, \text{MicroF}=5/8$

检索的评价

更多的评价准则：NDCG

(Normalized Discounted Cumulative Gain, 归一化折损累积增益)

- 针对非二值相关情况下的指标
- 每个文档不仅仅只有**相关**和**不相关**两种情况，而是有**相关度级别**，比如**0, 1, 2, 3**。

我们可以假设，对于返回结果：

- 相关度级别越高的结果**越多越好**
- 相关度级别越高的结果**越靠前越好**

$$\text{NDCG}(Q, k) = \frac{1}{|Q|} \sum_{j=1}^{|Q|} z_{kj} \sum_{m=1}^k \frac{2^{R(j,m)} - 1}{\log_2(1 + m)}$$

- $R(j, m)$ 是**评价人员给出的文档d对查询j的相关性得分**， z_{kj} 是归一化因子，保证对完美系统NDCG的值为1， m 是返回文档的位置

信息检索评价小结

- $P\backslash R\backslash F$
- P-R曲线
- AP
- Precision@N
- MAP
- NDCG

NeuHub
AI Links All

总体回顾

- 信息检索概述
- 布尔检索及倒排索引
- 信息检索模型综述
- 索引压缩与VB编码
- 查询优化与WAND算法
- 信息检索系统评估

NeuHub
AI Links All

一些可选的练习

- VB编码和解码：思考应用场景

Encoding

```
1: function ENCODE( $x$ )
2:   while  $x \geq 128$  do
3:     WRITE( $x \bmod 128$ )
4:      $x = x \div 128$ 
5:   end while
6:   WRITE( $x + 128$ )
7: end function
```

Decoding

```
1: function DECODE(bytes)
2:    $x = 0, s = 0$ 
3:    $y = \text{READBYTE}(\text{bytes})$ 
4:   while  $y < 128$  do
5:      $x = x \wedge (y \ll s)$ 
6:      $s = s + 7$ 
7:      $y = \text{READBYTE}(\text{bytes})$ 
8:   end while
9:    $x = x \wedge ((y - 128) \ll s)$ 
10:  return  $x$ 
11: end function
```

一些可选的练习

- WAND算法：深刻理解背后的思想

Given Q, k and the postings lists $L[0 \dots |Q| - 1]$ with:

$L[i].max \hat{=}$ the **maximum contribution** of the list

$L[i].cur \hat{=}$ the current element of the list

```

1: function WAND( $Q, k, L[0 \dots |Q| - 1]$ )
2:    $TopDocs = \emptyset$  ▷ Min Heap of size  $k$ 
3:   Threshold  $\Theta = 0$  ▷ Smallest score in  $TopDocs$ 
4:   while Not all lists are processed do
5:     Sort  $L$  based on  $L[i].cur$ 
6:     Select pivot list  $p$  such that  $\sum_{i=0}^{p-1} L[i].max \geq \Theta$ 
7:     Forward all lists  $L[0 \dots |p| - 1]$  to  $d_p = L[p].cur$ 
8:     Compute  $S_{Q, d_p}^{BM25}$  and insert into  $TopDocs$  if score  $> \Theta$ 
9:      $\Theta = \min(TopDocs)$  or 0 if  $|TopDocs| < k$ .
10:  end while
11:  Return  $TopDocs$ 
12: end function
  
```

一些可选的练习

- 信息检索评价指标的计算
 - AP
 - Precision@N
 - MAP
 - NDCG

NeuHub
AI Links All